IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 4, 2020, accepted February 21, 2020, date of publication March 5, 2020, date of current version July 2, 2020.

*Digital Object Identifier 10.1109/ACCESS.2020.2978632*

# An Optimized Frame-Driven Routing Algorithm for Reconfigurable SRAM-Based FPGAs

**LUDOVICA BOZZOLI, (Student Member, IEEE), AND LUCA STERPONE, (Member, IEEE)**
Dipartimento di Automatica e Informatica, Politecnico di Torino, 10129 Turin, Italy
Corresponding author: Luca Sterpone (luca.sterpone@polito.it)

**ABSTRACT** Reconfigurable SRAM-based Field Programmable Gate Arrays (FPGAs) are everyday more attractive due to their high integration, performance, flexibility, and upgradability. Run-time reconfiguration improves the reconfigurable computing paradigm allowing to rewrite just a portion of the FPGA configuration memory on-line. This enhances the flexibility and provides opportunities for new high-performing architectures able to adjust in-flight the hardware to the current payload. However, the performance of reconfigurable architectures is bounded by the efficiency of the reconfiguration procedure, which in turn is bounded by the amount of configuration frames to be rewritten in the memory. Furthermore, the lack of tools and design software to implement optimized reconfigurable architectures makes their performance less efficient than expectation. In this work, we propose an approach to enhance the performance of reconfigurable systems by reducing the reconfiguration time of reconfigurable resources. Our method is based on a frame-driven routing algorithm able to drastically reduce the number of configuration memory frames used in the design. We evaluate the optimization achieved with our algorithm on several benchmark circuits of different size and we investigate the performance and the routability for different placement solutions. Experimental results confirm that our approach reduces the reconfiguration time up to 40% with respect to traditional reconfiguration approaches for a wide range of circuits.

**INDEX TERMS** Frame, partial reconfiguration, reconfiguration time, routing algorithm, SRAM-based FPGAs.

## I. INTRODUCTION

Reconfigurable computing is increasingly gaining ground with respect to the classical hard-coded and programmable systems. In fact, hard-coded designs provide high performance at the cost of the flexibility, since the hardware data-path is specific to the application but fixed at the production stage. In programmable systems, instead, software algorithms can be freely implemented after production on a fixed general-purpose hardware, providing high flexibility, this time at the cost of performance. Reconfigurable computing represents a *synthesis* paradigm combining the flexibility of the software with the performance of the hardware. In fact, in reconfigurable systems both data-path and algorithms can be customized on the same support device, even after deployment [1]. For this reason, the last decade has seen a sharp increase in the usage of SRAM-based Field Programmable Gate Arrays (FPGAs) to implement High Performance Reconfigurable Computing (HPRC) systems [2], [3]. In fact, due to the possibility to modify or rewrite after the start-up their configuration settings, FPGAs represent the golden devices for implementing reconfigurable applications.

The possibility to access after the deployment the FPGA configuration memory, which controls the application functionality, makes them flexible and provides opportunities for innovative, upgradable, and robust architectural solutions for a wide range of applications [4], [5].

When just a portion of data-path is modified at run-time, we talk about Partial Reconfiguration (PR). This feature allows the user to select during the design phase the area of the device that can be modified after deployment and to perform such procedure without stopping the application [6], [7]. This allows several optimizations on the system performance, enabling time-space partitioning, area saving and power reduction techniques.

The associate editor coordinating the review of this manuscript and approving it for publication was Bijoy Chand Chatterjee.

Unfortunately, the performance of the reconfigurable application itself is bounded by the reconfiguration time. In fact, the time needed to perform this procedure is still high and depends mainly from two components: the amount of data to be rewritten inside the configuration memory and the interface used for accessing the memory.

The growing interest in partially reconfigurable systems, leads to the development of several configuration interfaces devoted to efficiently load the partial configurations from the off-chip non-volatile memory to the FPGA's configuration memory. Some examples are the slower Serial Configuration Mode and JTAG Partial Reconfiguration, and the faster Parallel Port Configuration Interfaces and the Xilinx Internal Configuration Access Port (ICAP) [8]. Even if several optimizations of such interfaces have been investigated (e.g., [9]–[11]), few solutions have been suggested to act on the amount of data to be rewritten in the configuration memory. In details, the atomic addressable configuration memory unit for Xilinx 7 Series FPGA is the frame, a 3,232 bits word [8]. This represents a strong architectural limit for fast reconfiguration, considering that the time needed for a frame memory transfer is of hundreds of microseconds ($\mu$s).

A recent approach proposed an implementation flow to reduce the configuration time by reducing the area of the implemented circuit [12], thus directly reducing the overall amount of configuration data. Anyway, the optimal placement solution provided by the vendor tool is not considered anymore, leading to potential side-effects, such as routing congestion. Furthermore, this approach could result critical or unfeasible for large designs.

In [13], we introduced a preliminary approach to reduce the amount configuration memory data of a circuit mapped on a SRAM-based FPGAs without modifying its topology. We achieve this result only acting on routing resources without modifying the optimal placement solution provided by commercial tools. In fact, after an in-depth analysis of the routing structure, we were able to realize a routing algorithm, FeDRA (Frame-driven Routing Algorithm) that is able to route circuits reducing the number of used frames and thus reducing the configuration time up to the 40%.

In this paper, we present and evaluate the FeDRA algorithm applying its optimization to larger circuits and analyzing the performance of our algorithm from the point of view of the execution time and routability with different area constraints. Experimental results demonstrate that FeDRA can successfully implement optimized designs also for larger circuits, under tight area constraints, and when different designs are placed in the same reconfigurable area.

### A. THE MAIN CONTRIBUTION

The main contribution of our optimization technique is represented by its efficiency coupled with its non-intrusiveness in the system performance and in standard reconfigurable design process. In fact, our solution doesn't need any different configuration interface and it is applied only at the routing stage, leaving all the other steps of the application development executed by vendor tools. Additionally, to the best of our knowledge, it is the first FPGA routing algorithm exploiting the awareness of the link between configuration memory organization and configurable resources.

The main intuition behind this work is that evaluating resources from the point of view of their configuration settings rather than from their topological position allows to perform the optimization on the reconfiguration time without renouncing to the optimal and already optimized placement provided by the vendor tool. Additionally, it scales efficiently with the congestion of the circuits and does not introduce any relevant overhead in the other performance parameters, such as resources and time overhead. The rest of the paper is organized as follows: in Section II and III the background and the state of the art on the FPGA fabric, configuration memory, reconfiguration procedures and implementation flow are provided; Section IV describes the FeDRA algorithm; Section V presents the experimental results while conclusions and future works are discussed in Section VI.

## II. BACKGROUND

### A. RECONFIGURABLE FPGA ARCHITECTURE

In general, reconfigurable hardware provides high flexibility and upgradability since its functionality can be customized to obtain a variety of applications. Although several reconfigurable architectures have been proposed in the past, FPGA represents the most consolidated solution. The characterizing feature of the FPGA is to act as a blank hardware where any design can be tailored, providing on a unique device the performance of the hardware and the flexibility of the software. This duality relies on a two-layer fabric: one layer of configurable resources on the top a configuration memory layer.

The resource layer consists of a regular array of basic configurable blocks containing logic and routing resources. Logic is devoted to implement logical and arithmetic operations and typically consists of Look-up Tables (LUTs) storing the truth table of the wanted function and Flip-Flops (FF) as storage element for sequential behavior. Programming bits define LUTs truth tables and FFs activation.

Connections among logic nodes are implemented with programmable routing to build the interconnect structure.

As for the logic, each switch point in the connecting tissue is programmed by few bits: programmable switches are configured to switch among different logic block inputs/outputs, vertical and horizontal junctions, and different hierarchy levels of segment length. The configuration layer consists of an array of memory cells storing the configuration settings for programmable resources. These binary values determine whether routing segments are activated to connect nodes and the functionality of specific logic resources.

The collection of data inside the configuration memory univocally defines a specific circuit for a given hardware task. The procedure of writing the binary information inside the configuration memory is called reconfiguration. A first

configuration is loaded inside the memory at the start-up; when configuration data are written at run time, the procedure is called Dynamic Reconfiguration while Partial Reconfiguration consists in writing only a subset of configuration data at runtime.

The most flexible configuration storage in commercially available FPGAs are volatile SRAM cells, which allow fast and repeatable reconfiguration in a well-known technology.

Nowadays, among the others, Xilinx SRAM FPGAs result preferable for implementing reconfigurable applications. In fact, with respect to other vendors counterpart, Xilinx provides more powerful and documented CAD tools, higher working frequency and its architecture results more suitable for modular partial reconfiguration [14], [15].

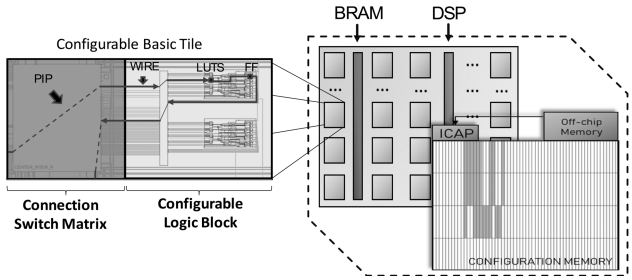


**FIGURE 1. Overview of the current FPGA architecture: on the left the basic Tile unit is shown, with detail of the programmable resources and their functions; on the right the two layers fabric is schematized with the configuration memory layer and the programmable resource layer integrated with hardwired resources.**

## III. THE XILINX 7-SERIES ARCHITECTURE

Each basic programmable unit of the 7 Series Xilinx FPGA is made by two parts: the Control Logic Block (CLB) that contains the programmable logic, and the Switch Matrix (SM) that contains the programmable interconnections. The programmable logic consists of LUTs and FFs [16].

Programmed LUTs functions and activated FFs implement the logic nodes of the programmed design. In order to connect the inputs and outputs of different CLBs, the programmable routing resources inside the switch matrices are used. The programmable routing resources are called Programmable Interconnect Points (PIPs) and make the configurable link between a switch matrix input and a switch matrix output. Each output of the switch matrix is connected to a hardwired segment, which is connected to the input of another switch matrix or to inputs or outputs of a CLB. The collection of PIPs and wires builds the nets that route the circuit.

Additionally, following the trend of providing more integration, in the current FPGA architecture the programmable resources are interleaved to hardwired components such as integrated microprocessors, Digital Signal Processing (DSP) units and memory blocks to implement user memories (BRAMs). This is done to provide to the users specialized hardware blocks for implementing functionalities that could be not optimized or resource-consuming when implemented in programmable logic. A formalization of the current architecture is provided in Figure 1.

The Tiles are grouped in Clock Regions, sub-portions of the FPGA fed by the same Global Clock Line (GCL) to properly manage the clock distribution.

The behavior of the programmable resources is defined by the content of the configuration memory layer. The configuration memory content is a binary file, called bitstream, generated after synthesis and implementation. The bitstream is organized in a regular structure of various bit arrays spanning in vertical the whole device area. This structure is called frame and it is the smallest addressable configuration memory segment for reconfiguration [8].

The regular organization of the resources inside the FPGA array is somehow translated in the configuration memory structure, which shows the same regularity. With our tool COMET [21], we observed that the frames relative to Tiles belonging to the same Clock Region are stored sequentially. Anyway, the order in which the bitstream portions relative to different Clock Regions are stored is different with respect to the coordinates provided by Xilinx Vivado Tool and this order changes for each device model.

Fig. 2 shows how the stream of bits in the bitstream is grouped in frames and how these frames are mapped with respect to the programmable resource layer for the Kintex-7 FPGA. In fact, to link a sub-portion of the bitstream to the FPGA region it describes, the following steps need to be performed: all the bits must be grouped in frames of 3,232 bits; this collection of frames must be divided in subsections, each one of them relative to a given Clock Region; finally, such regions must be reordered to match the FPGA structure.

Typically, the top half of the Clock Regions (according to the device view) is stored in reverse order in the first half of the bitstream. The bottom half Clock Regions appear in the second half of the bitstream in the same order of the device, as shown in Fig. 2.

For the Xilinx Kintex-7 XC7K325T FPGA, the array of Tiles is organized in a matrix of $50 \times 350$. The configuration memory instead can be seen as a frame matrix 3, $240 \times 7$. In fact, each column of Tiles of every Clock Region is programmed by bits that belong to the same frame and the position of the bits controlling the storage, the logic elements and interconnection resources is periodic inside the configuration memory.

The entire Tile is described by a sub-matrix $36 \times 64$ of configuration bits. Each column of 64 bits along the y axes belongs to the same frame. In other words, each frame (partially) configures resources belonging to 50 different Tiles along the vertical axes within a Clock Region.

In Figure 3.a two contiguous Tiles are shown. Figure 3.b shows their correspondence with configuration memory frames: in details, each CLB is configured by 10 of the 36 sub-portions of frames, while the remaining 26 program the routing resources. In Figure 3.c is reported the bitmap of a portion of configuration memory relative to the same portion of the device. The image has been obtained with COMET on

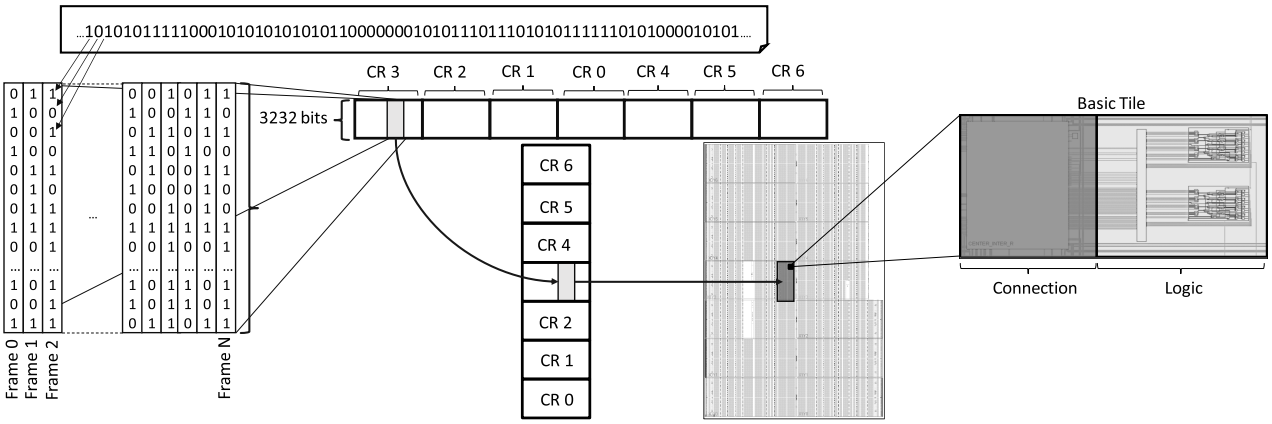


**FIGURE 2.** An overview of the Bitstream logic organization for Xilinx 7 Series FPGA. This example is relative to the Kintex-7 FPGA.

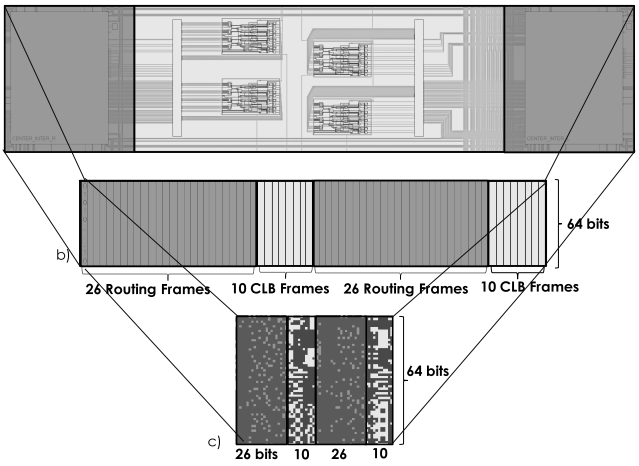


**FIGURE 3.** Xilinx Kintex-7 Tiles topology (a) and configuration memory correspondence with configuration memory frames (b) and bits (c).

a sample design programmed on the Kintex-7. Each column of pixels represents a portion of 64 bits of a frame and the white pixels represent the configuration memory bits set to 1. It is also possible to see the different bits distribution in the part devoted to program the routing and the CLB sides [21].

### A. RECONFIGURATION TIME AND TEQUINQUES

The feature of accessing FPGA configuration memory after deployment can be exploited in different ways and for different purposes. Before defining them, a general formal definition of reconfiguration time $t_{rec}$ is provided in (1):

$$t_{rec} = A_{atomic\ memory\ unit} \cdot N_{bits\ per\ unit} \cdot t_{download\ bit} \quad (1)$$

The $A_{\text{atomic memory unit}}$ is the amount of basic addressable memory segments to be reconfigured and it is related to the implemented circuit; $N_{\text{bits per unit}}$ is the data size of the atomic addressable memory unit and it is a term related to the architecture of the target reconfigurable device; $t_{\text{download bit}}$ is the average time for each bit memory transfer and it depends from the used configuration interface.

For Xilinx 7 Family the atomic memory unit is the frame, which is a 3,232 bits word, and the time needed for downloading each frame through the ICAP is around 100 $\mu$s.

Considering the reconfiguration time definition in (1) is clear that what can be optimized without re-thinking the architecture or the configuration interface is the number of used frames, acting at the design level.

In Fig. 4.a, an example of configuration memory usage for a given design is provided, followed by the typical Partial Reconfiguration techniques.

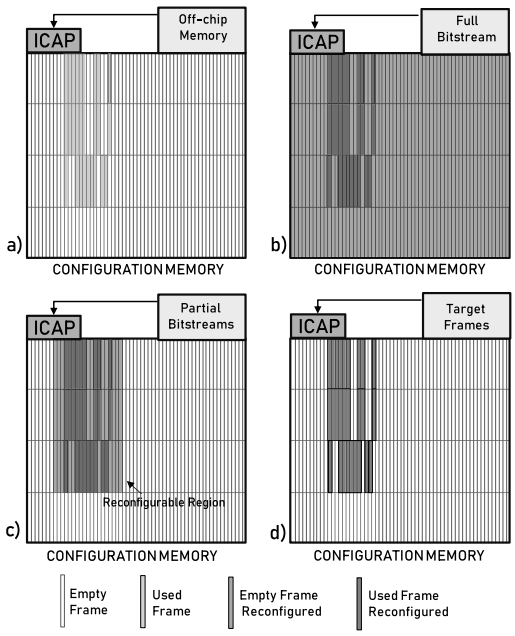


**FIGURE 4.** Overview of the typical Reconfiguration Techniques.

The used frames can be interpreted as frames to be re-written to refresh their content or as frames to be modified to change the circuit behavior. The Scrubbing, in Figure 4.b is typically used when the whole content of the FPGA configuration memory needs to be rewritten; in this situation, all the configuration memory frames are reloaded, thus the off-chip

memory from which configuration data are read contains the full bitstream. This technique is mostly used in aerospace applications, where radiation effects can cause bitflips inside the memory cells: by re-writing the correct content inside the cells is possible to avoid or recover malfunctioning [3], [18], [19]. Anyway, the typical partial reconfiguration approach, shown in Figure 4.c, consists in configuring only the configuration frames belonging to the target Reconfigurable Region (RR) that contains the design that should be modified on-line. This technique relies on defining during the development phase the area that will contain resources to be dynamically allocated (or refreshed) during the device mission and producing a-priori their relative partial bitstreams [6], [10], [20]. In this case, the off-chip memory contains all the partial bitstreams relative to the possible configurations of the target reconfigurable area. In both the techniques all the configuration frames are transmitted to the FPGA's configuration memory independently from their content: thus, even empty frames (i.e., not programmed) are written into the configuration memory. The best approach in terms of reconfiguration time is the Capillary one, shown in Figure 4.d, that has been proposed [21] in order to download exclusively the used frames. This technique requires more effort during the development process, but truly minimizes the reconfiguration time since it minimizes the number of frames to be reconfigured.

## IV. STATE OF THE ART

In order to obtain the FPGA implementation of a circuit from its Hardware Description Language (HDL) design, the following steps need to be performed within the vendor tool: Synthesis, Technology Mapping, Placement, Routing and Bitstream Generation. Even if FPGA design development process follows conceptually the same steps of ASIC development process, each phase has a different meaning. The Synthesis and the Technology Mapping steps translate the HDL description to a gate level description of the circuit that uses the functional blocks of the target technology: within the FPGA these blocks are LUTs, FFs, In/Out pins, BRAMs and DSPs. The Placement assigns the abstract functional blocks identified in the previous steps to the physical resources of the target FPGA while in the Routing stage the routing segments to connect the logic blocks that produce the wanted connections among logic nodes are identified. Typically, these two last steps are referred as a unique step called Implementation (or Place&Route, P&R) since they are strictly related. Once the Implementation of the design is completed, the configuration data to be stored inside the configuration memory describing the target implemented circuit are coded into the Bitstream binary file.

Considering that FPGA fabric has a finite amount of programmable resources, the P&R design stage is more critical since, differently from ASIC, it is constrained by the effective availability of the resources. Additionally, in FPGAs, the performance, and in some cases even the feasibility, of the implementation is strongly dependent from the quality of P&R solution and the trade-off among resources utilization, power end delay results even more difficult to reach due to the lower degree of freedom. For this reason, FPGAs placement and routing algorithms have been a crucial point of research both in literature and in practice.

Routing algorithms can be classified according to the algorithmic approach and to their cost function. From the algorithmic point of view, the geometric routing approach has been individuated as one of the most effective for FPGA routing [22]. Among this category of algorithms, the most used are the Maze Router, the A$*$ Search and the Pathfinder [23]. When the routing constraints are tight, the Boolean-based algorithms are preferred, since they present the advantage of considering nets simultaneously [24].

In addition to the primary goal, which is to successfully complete the routing of the circuit, routing algorithms typically evaluate their decisions also to minimize or maximize a given parameter or to satisfy conditions. This secondary goal can be defined as cost function or as routing policy.

FPGA routing algorithms typical policies are the minimization of the delay, minimization of the power consumption, mitigation of congestion or reliability enhancement. Some examples for these categories are respectively the TRACER_fpga [25], the algorithm proposed by Roy [1], CeRA [26] and RoRA [27].

Cost functions can also try to satisfy more than one of these parameters at the same time, anyway the final outcome is always a trade-off. For instance, TRACER minimizes the circuit delay by minimizing the path length, but at the same time it has a long execution time and does not consider routing density. Power consumption minimization typically happens at the cost of the area, such as the reliability enhancements. Above all, all these algorithms are by definition computationally intensive and typically run in complex Computer Aided Design (CAD) toolchains for FPGA development.

Anyway, if the routing problem is complex, the placement is even more crucial. Considering that Placement typically has the greatest impact on overall design performance, the Routing solution is strongly dependent from the quality of the Placement: a poor Placement can lead to weak performance both in terms of power consumption and delay, and in the worst case, even to the unfeasibility of the routing [28]. Furthermore, differently from Routing, the exhaustive exploration of the placement solutions space and the achievements in obtaining effective heuristic placement algorithms represent still open problems [29].

When Partial Reconfiguration is involved, the development flow becomes even more complex: reconfigurable and static areas must be identified at the beginning of the process; the interfaces between static and reconfigurable areas should be defined in such a way that the requirements of different circuits to be allocated in the same dynamic area are satisfied; hardware modules for dynamic areas should be synthetized and implemented in a separate flow with respect to static circuitry in order to create all the partial bitstreams to be used at run-time.

Beside the optimizations performed during the development process, when reconfiguration is involved in the computation, the procedure itself can introduce a huge overhead in the application, nullifying the delay and power saving achievement obtained in the implementation step on the circuit. Considering that in some cases the time consumed by the reconfiguration procedure represents a huge portion of the application execution time, it is essential to minimize its overhead to maximize the overall performance of the system. Typically, this is performed exclusively in the application layer, playing with hardware tasks scheduling [30]–[32]. Anyway, to consider the reconfiguration time as optimization goal even during the design implementation stage can truly enhance the overall system performance.

To the best of our knowledge, the unique work in literature following this approach is [12], where Vansteenkiste *et al.* suggested a Place & Route tool, called TPaR for enhancing the performance of FPGAs dynamic reconfiguration. This work targets the optimization of reconfigurable applications in the implementation stage. Its goal is to reduce the area occupied by the reconfigurable circuit and thus the time overhead needed for the update of the relative configuration memory portion. Additionally, it represents one of the few works aimed to fill the lack of automatic tools for the development of reconfigurable applications.

Anyway, considering the complexity and the cruciality of placement and the efficacy of vendor placer in performing delay, wire length and congestion optimizations, to discard the vendor optimal placement solution could result in a loss of performance. Additionally, when the amount of resources used by the design increases, it becomes more difficult (if not impossible) to play the optimization proposed in [12].

The proposed FeDRA algorithm is a geometric-based algorithm based on the reconfiguration time minimization cost function. This is done selecting routing resources that introduce the minimum overhead in terms of configuration settings to be updated at runtime. In order to achieve such optimization without penalty in other performance parameter such as delay, congestion and power consumption, FeDRA has been developed to be integrated within the vendor tool and to start from the vendor optimal Placement solution. In fact, considering that routing represents up to 70% of FPGAs resources, it is reasonable to perform the frame optimization on it without interfering with the placement of logic nodes, which is already optimized from the vendor tool and typically performed looking ahead to the optimal global routing.

## V. THE FRAME DRIVEN ROUTIN ALGORITHM: FeDRA

The main idea behind our approach is to reduce the reconfiguration time of the partial reconfiguration process on SRAM-based FPGAs by exploiting a frame routing policy in order to reroute FPGA connections using the minimum number of frames. In this section, we first summarize the bitstream and frames decoding performed. Then, we describe the key concept, which relies on an in-depth analysis of the bitstream with respect to the routing resources. Later, we describe the FeDRA approach, presenting the main algorithm routines.

### A. BITSTREAM ANALYSIS AND FRAME DECODING

As is it possible to perceive, the optimization of the configuration procedure by minimizing the number of frames used to implement a circuit requires a detailed knowledge of the bitstream encoding. In addition to the current trend of the digital design to increase the abstraction level, the lack of information from the vendors about the details of their FPGA fabric is still high. The first part of the study consists in an in-depth investigation and taxonomy of the Family 7 routing resources. Their organization can be retrieved from their architectural routing format [33]. PIP segments are identified by their direction with respect to the hardwired segments they are connecting and the coordinates of the switch matrix they belong on the FPGA.

To build the database, we extracted all the available Programmable Interconnect Points. The time to process the whole extracted PIP list could be enormous and the list contains several replicas of the same primitive resource relative to different Tiles in the FPGA.

For this reason, the Essential PIPs have been pruned (21,081 Essential PIPs on 124 millions PIPs for the Kintex-7). To find correlation between PIPs and configuration memory bit coordinates, we realized a TCL script executed inside the vendor tool, which sequentially programs the FPGA with one Essential PIP and gives back its Frame Address Register (FAR) [8]. Once the coordinates of the Essential PIPs are individuated, all the others can be easily retrieved due to the high regularity of the FPGA [13]. Finally, we used COMET tool to verify and formalize the information obtained with this methodology [17].

After this process we were able to link each routing resource inside the FPGA to the number and the position of the frames involved in its configuration.

### B. ROUTING POLICY

After a close examination of the bitstream database we were able to notice that routing resources are programmed by a variable number of bits within a variable number of frames. Furthermore, often happens that the bits used to configure resources on the same Tile column belong to the same frames, as it is shown in the examples in Figure 5.

Typically, the number of bits used for each routing resource is not fixed. Each PIP is programmed by bits belonging from 1 up to 4 different frames crossing a Switch Matrix. Since each frame spans in vertical on 50 Tiles, a routing resource can be programmed by frames common to resources in the same switch matrix such as in other switch matrices in the same column [17].

Starting from this key concept, it is possible to reduce the number of total frames by locally changing the routing of a net (signal driver from the output of the CLB to the input of one or more CLBs), this without any change in the original (and optimal) placement from the vendor tool.

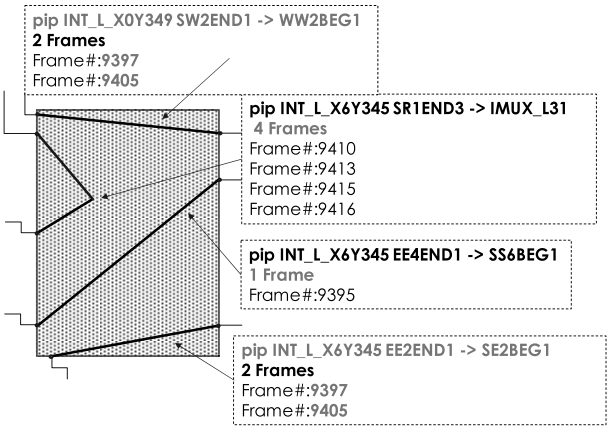


**FIGURE 5.** The three main types of Programmable Interconnect Points (PIPs) of a routing switch matrix: 1, 2 or 4 programmed frames.

In fact, in each Switch Matrix several PIPs exist to cover the same distance and direction. Each Switch Matrix Junction has in average 32 PIPs that start/insist on it. These PIPs share the same driver and among them there is more than one PIP going in a given direction. During the routing process the algorithm selects among them the one that minimizes the overall number of frames and at the same time properly routes a net, respecting the signal integrity.

Even if the FeDRA algorithm performs the same computation, two optimizations are possible:

- *Relative Routing Policy*: among all the available PIPs with the same behavior and considering the used frames in the same column of tiles, the PIP that introduces less additional frames is selected.
- *Absolute Routing Policy*: if no matching with already used frames exists, among all the available PIPs with the same behavior, the one that is programmed by the smallest number of frames is selected.

In Figure 6 and 7, an example of both Optimization Policies on a sample net is reported: the net in Figure 6 is the Standard one, routed by the vendor tool, while in Figure 7 the one routed using the Frame-Driven Algorithm.

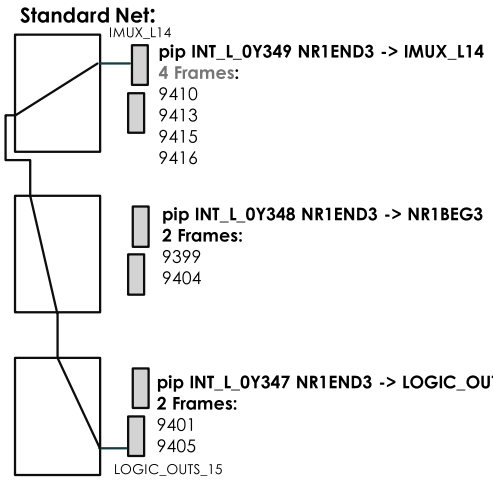


**FIGURE 6.** An example of standard net routed by the vendor tool showing the sequence of PIPs used without considering the frame reference overlapping.

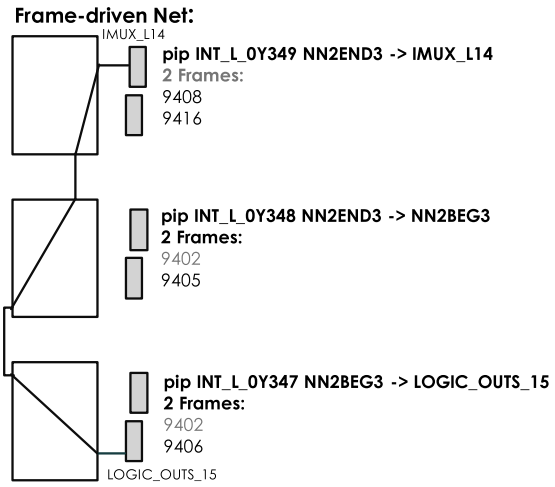


**FIGURE 7.** The net of the example of Fig. 6 after the FeDRA optimization: by means of the absolute and relative policies 3 frames have been saved.

Both the nets take signals for the Output 15 of the bottom CLB to Input 14 of the top CLB crossing 3 Switch Matrices. To implement the net in the standard way 8 frames are used, 2 for the first PIP (starting from the bottom), other two for the second one and 4 for the last one, and no one of them is shared between resources.

In the net of Fig.7, obtained with FeDRA, one frame is saved with the routing of the first two PIPs according to the R*elative Policy*, since Frame #9402 is shared by the two resources.

Other two frames are saved in the last step, since a resource that needs just 2 frames instead of 4 is used. Thus, it is possible to see that with our approach 3 of 8 frames are saved.

### C. FeDRA ROUTING ALGORITHM

The goal of FeDRA is to find a complete routing solution for a circuit and at the same time to minimize the number of used frames. The algorithm performs its computation starting from the Vivado placement solution of the circuit, which consists in a description of how logic nodes and In/Out pins are connected. The placement information extracted with a TCL script from Vivado, is used as constraint for the optimized circuit and coded in a format suitable for the Frame-driven Router that re-shapes all the nets connecting placed nodes to perform the frame saving. In Figure 8 a simplified version of the algorithm is reported. At the beginning of the execution, the routing solution for the optimal Vivado placement is empty. The net information obtained from the vendor tool is iteratively extracted until all the nets of the circuit have been routed.

The path connecting the source and the drain/s of each net can be decomposed in several steps, individuating at each iteration a temporary source and a temporary drain that could be connected through a single PIP. For each couple of temporary

```
//Initialization Phase
READ Vivado_Placement_Solution
Routing_Solution = {EMPTY}
Current_Design_Frame_Vector = {EMPTY}

//Net Extraction
While (NET_DATABASE != {EMPTY}){
    READ NET
    while(NET != {ROUTED}){
      SET Source_tmp
      SET Drain_tmp
      //Frame Filtering
      for each PIP(Source_tmp; Drain_tmp) in PIP_DATABASE{
          cost_new = compute_cost_function()
          if (cost_new < cost_tmp){
            cost_tmp = cost_new
            PIP_best = PIP
            next_DFV = tmp_Design_Frame_Vector
          }
      }
      //Update
      ADD PIP_best to NET
      Current_Design_Frame_Vector = next_DVF
    }
    ADD NET to Routing_Solution
}
Routing_Solution = {COMPLETE}
```

**FIGURE 8.** The FeDRA algorithm pseudo-code.

sources and drains a subset of PIPs exists that satisfies the connectivity requirements. For all the PIPs available on this subset, a cost function is computed and evaluated in order to select the resource that minimizes the overall number of frames (Fig. 9).

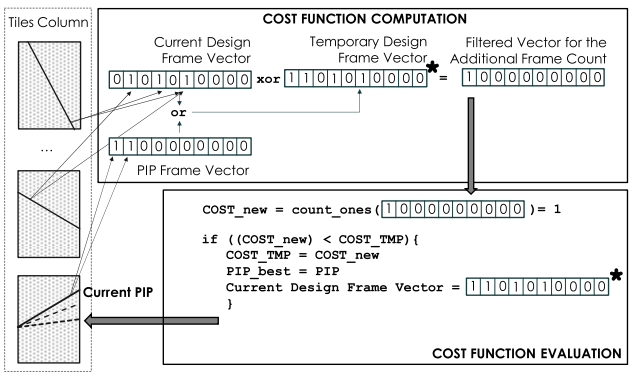


**FIGURE 9.** An example of cost function computation and evaluation performed by the FeDRA algorithm.

The computation is based on a very long flag vector, the Current Design Frame Vector, which traces the frames already used in the FPGA (1 in the corresponding position if used, and 0 if empty). Every time a new PIP is evaluated, the additional frames potentially introduced to program it are interpolated through simple bitwise logic operations with the Current Design Frame Vector.

In fact, the bitwise or between the PIP Frame Vector relative to the PIP under evaluation and the Current Design Frame Vector returns the overall frame usage if the resource is used in the routing solution. The bitwise xor among the so obtained temporary vector and the original one provides the absolute number of additional frames introduced by the resource, which is used as cost function.

If the obtained cost in terms of frame overhead is smaller than the present one, the cost is updated, the current PIP is saved as the temporary optimal one.

When all the available PIPs have been processed, the optimal one is added to the routing solution, the Current Design Frame Vector is updated according to this choice and the temporary drain becomes the new temporary source for the next step. This procedure is repeated until the whole net is routed and so on, until the whole circuit rerouting is complete. A more detailed description could be found in [13].

## VI. EXPERIMENTAL RESULTS

The FeDRA algorithm has been developed as a software tool integrated with the Xilinx Vivado IDE 2017.2. It can elaborate synthesized designs and bitstreams of the Xilinx 7 Family FPGA. In [13] we evaluate our algorithm for 5 different benchmark circuits implemented in Vivado in standard condition (i.e., with the board frequency as timing constraint and without area constraint) proving that FeDRA can successfully re-route and save from more than 35% up to the 41% of reconfiguration time with respect to the vendor tool implementation. This evaluation has been made implementing the five different benchmark circuits on the Kintex-7 XC7K325T device and comparing the routing solutions obtained with Vivado Place and Route tool with the one obtained substituting the routing step with FeDRA. In this work, we extend this evaluation by applying FeDRA optimization to larger circuits and introducing different placement constraints. At the same time, we evaluate the performance of our router in terms of execution times and the routability, analyzing the obtained solution with respect to vertical and horizontal long lines and short segments.

In detail, benchmark circuits have been implemented both with the Xilinx Vivado tool flow and by the routing stage with FeDRA.

To obtain circuits with congested area we added area constraints in the placement stage. After the implementation was completed, the Configuration memory bitstreams have been produced for the two versions. The bitstreams have been loaded on the target device for the evaluation of the reconfiguration time [13].

The frames usage comparison among the original and the optimized versions has been obtained with the Frame Analyzer, an ad-hoc software developed for the purpose. The Frame Analyzer takes the bitstream as input and returns the number of frames programmed (i.e., with at least one bit set to 1). For each benchmark a report on FeDRA routing solution and execution time has been produced and evaluated. The evaluation framework used for the analysis is reported in Figure 10.

This allows us to observe a positive trend on the optimization achievable with respect to the size of the circuits, the occupied area and its congestion, and to characterize the performance of our router in terms of routing solutions and execution times.

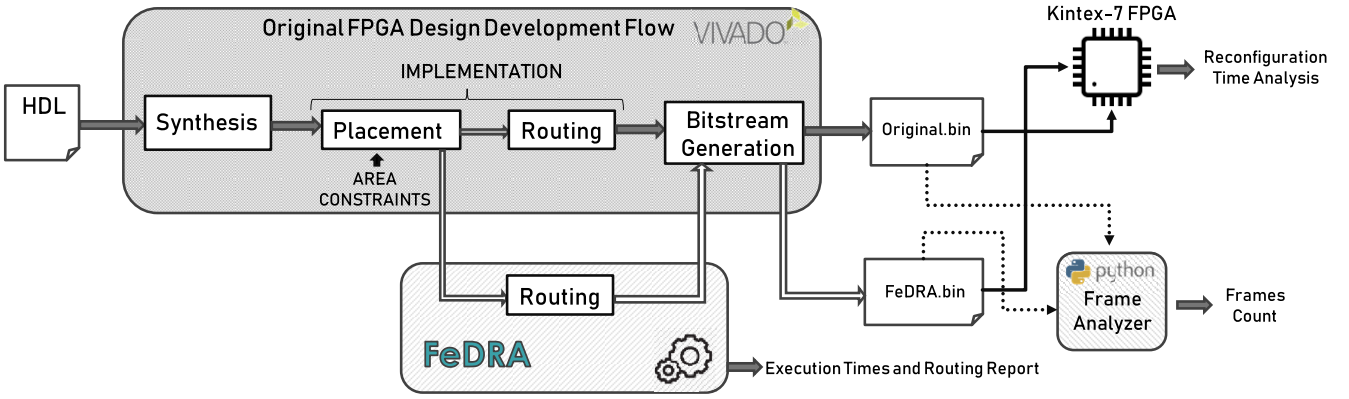


**FIGURE 10.** FeDRA Evaluation framework: the comparison among designs obtained with the original development flow and the one which integrates FeDRA is performed on the target devices for reconfiguration times and with the Frame Analyzer software for the frame usage count. The execution times and the routability analysis are extracted from the FeDRA reports.

### A. BANCHMARK DESCRIPTION AND EVALUATION IN STANDARD CONDITIONS

The evaluation performed to prove the feasibility and the advantages of our solution has been done on nine circuits: seven ITC'99 benchmarks [34], a Cordic Core [35] and the miniMIPS [36] processor. The circuits have been implemented on a Kintex-7 XC7K325T SRAM-based FPGAs and the resources are detailed in Table 1. The circuits have been implemented using the nominal board frequency of 100 MHz and without inserting area constraints. In Table 2, we reported the frame usage comparison between the original circuit version and the one obtained by FeDRA. The results demonstrate an average optimization of the frame usage of 35%. Any impact is observable on the delay of the circuit and a minimal overhead in terms resources in standard condition. In fact, the PIPs overhead introduced by our routing policy it is negligible and it scales with the circuit size. In fact, the additional percentage of resources becomes almost infinitesimal for bigger circuits: it goes from 0.97% for b05 to the 0.16% for miniMIPS [13].

**TABLE 1.** Characteristics of benchmark circuits implemented with Xilinx Vivado 2017.02.

| Circuit | LUTs [#] | FFs [#] | Nets [#] | PIPs [#] | IOs [#] |
|---|---|---|---|---|---|
| b05 | 84 | 34 | 110 | 1,596 | 39 |
| b12 | 223 | 119 | 242 | 3,776 | 13 |
| Cordic_r2p | 949 | 1,001 | 1,120 | 29,163 | 74 |
| b14 | 2,123 | 219 | 2,103 | 29,083 | 88 |
| miniMIPS | 2,712 | 2,000 | 4,797 | 72,987 | 175 |
| b22 | 3,686 | 613 | 2,942 | 42,203 | 56 |
| b17 | 4,897 | 1,350 | 4,850 | 70,937 | 136 |
| b18 | 11,904 | 2,842 | 11,152 | 158,886 | 61 |
| b19 | 22,565 | 5,686 | 20,877 | 304,124 | 53 |

**TABLE 2.** Benchmark routing frame usage comparison between original and FeDRA routing algorithm in standard condition.

| Circuit | Original | FeDRA | Optimization |
|---|---|---|---|
| b05 | 183 | 113 | 38.3% |
| b12 | 237 | 145 | 38.8% |
| Cordic_r2p | 702 | 412 | 41.3% |
| b14 | 1,794 | 1,126 | 37.2% |
| miniMIPS | 2,542 | 1,645 | 35.3% |
| b22 | 1,595 | 1,016 | 36.3% |
| b17 | 2,910 | 1,844 | 36.6% |
| b18 | 4,514 | 2,890 | 36.0% |
| b19 | 7,005 | 4,555 | 35.0% |

Please notice that it is possible to observe a slight dependence between the optimization and the size of the circuits. In fact, we observed a higher optimization for smaller circuits with respect to the large ones. This is due to the different number of registers, In/Out ports and the heterogeneous number of Nets. For instance, the miniMIPS has a lower optimization versus b22 and b17 mainly because of the higher routing congestion.

### B. AREA CONSTRAINT AND ROUTABILITY ANALYSIS

In order to further analyze the performance characteristics of the proposed FeDRA algorithm, we perform the evaluation of the frame saving considering a restricted placement area. Please consider that this is the typical scenario for reducing the area overhead of implemented circuits. Considering this purpose, the benchmark circuits have been implemented forcing the placement in a limited FPGA area.

We reported in Table 3 the benchmark circuits compression metrics in terms of used resources and congestion percentage of the area. The slices required are the ones used by the circuit to implement its logic and memory elements while the available slices are the number of slices available for

**TABLE 3.** Circuit compression metrics.

| Circuit | Slices Required [#] | Slices Available [#] | Module Congestion [%] |
|---|---|---|---|
| b05 | 24 | 24 | 100 |
| b12 | 56 | 56 | 100 |
| Cordic_r2p | 291 | 300 | 97 |
| b14 | 241 | 252 | 95.63 |
| miniMIPS | 888 | 900 | 98.67 |
| b22 | 803 | 816 | 98.41 |
| b17 | 1,321 | 1,786 | 73.96 |
| b18 | 2,899 | 3,096 | 93.64 |
| b19 | 5,953 | 6,392 | 93.13 |

the considered placement area. The module congestion is relative to the utilization of the available slices. Please note that in some cases the utilization is lower (e.g., benchmark circuit b17) due to the higher usage of memory slices related to the higher demand of In/Out ports and memory elements.

Figure 11 reports an example of unconstrained and constrained placement area for the benchmark circuit b18. In Table 4, the frames used by the benchmark circuits implemented within the placement bounding box before and after the optimization are reported. We observed that the optimization slightly decreases with the size of the circuits, going from a maximum of 39.2% to a minimum of 32.2% for the biggest circuit. We show in the graph of Figure 12 the comparison

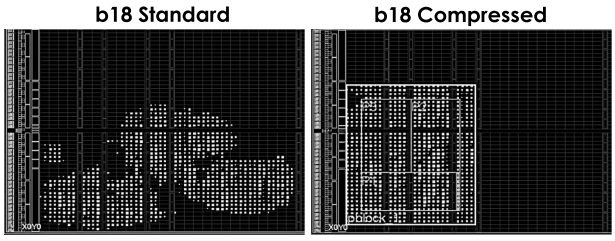


**FIGURE 11.** Standard (left) and compressed (right) placement solutions for b18 benchmark.

**TABLE 4.** Benchmark routing frame usage comparison between original and FeDRA routing algorithm with high congestion.

| Circuit | Original | FeDRA | Optimization |
|---|---|---|---|
| b05 | 143 | 87 | 39.2% |
| b12 | 170 | 104 | 38.8% |
| Cordic_r2p | 335 | 208 | 37.9% |
| b14 | 442 | 291 | 34.2% |
| miniMIPS | 1,409 | 912 | 35.3% |
| b22 | 1,145 | 764 | 33.3% |
| b17 | 2,070 | 1,352 | 34.7% |
| b18 | 3,185 | 2,108 | 33.8% |
| b19 | 5,817 | 3,942 | 32.2% |

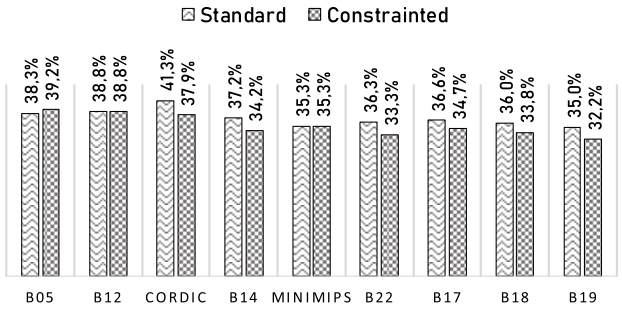


**FIGURE 12.** Frame Saving without Area constraints (Standard) and with Placement Area Constraints (Constrained).

between the frame saving obtained in standard condition and the one obtained with placement area constraints. It is possible to observe that the impact of the congestion is minimal and even positive in some cases. For example, in case of smaller circuits like b05, few resources give less optimization opportunities and a constrained area allows the overlap of more configuration frames. Vice versa, when the number of resources increases, we observe a negligible degradation of frame saving.

Dealing with the algorithm performance in terms of execution time, in Fig.13 we report as an example the result obtained for b12 for 6 different area constraints. To make this evaluation, we measured the execution time of the algorithm working with different area constraints, starting from the smallest one (1:1) and progressively relaxing the constraint along X axes to an area 8 times larger of the minimum one (8:1). As it is possible to observe, we have the longest execution times for the corner cases. In fact, we can see that the execution time trend is to decrease with the area reduction until the 1.5:1 constraint, and then to increase again. We performed a detailed analysis of the routing delay related to the different routing switches used by the circuits mapped on the FPGA. We perform a routability analysis in terms of short and long lines usage. The result of this analysis applied to the benchmark b12 is reported in Figure 14.

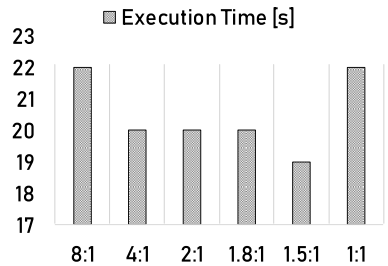


**FIGURE 13.** The proposed FeDRA algorithm execution times with different area constraints for the benchmark b12.

Thanks to this analysis, we observed that the Horizontal Long Lines quantity decreases with the horizontal area reduction, while the amount of Vertical Long Lines increases to compensate the Horizontal Lines and to avoid the usage of

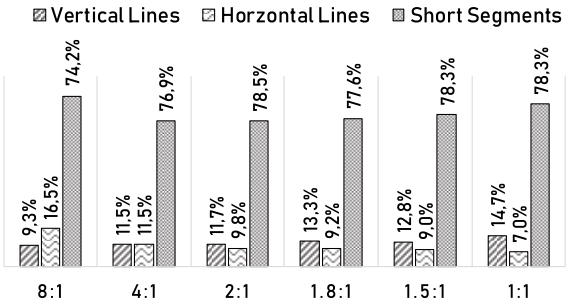


**FIGURE 14.** The Horizontal Long Lines, Vertical Long Lines and Short Segments ratio on the benchmark b12 for different area ratios.

enormous quantity of Short Segments, which require more routing time and add delay in the circuit.

On the other side, the Short Segments usage is high also for the 8:1 implementation (the circuit is larger, and resources are far from each other) and decreases for the 4:1 implementation.

This drop is due to the increased usage of shorter nets that are not introducing routing congestion. In fact, when we reduce further the circuit area, the number of Short Segments increases, this time due to the effect of congestion. Furthermore, we can attribute the long execution time of the bigger areas to the fact that the circuit is larger, nets are longer, and more resources need to be used. So, when the circuit shrinks, the time required to route it decreases. At the same time, as the circuit has harder area constraints, the congestion increases, requiring more resources and more time to be routed.

### C. RECONFIGURABLE MODULES ALLOCATION/ DEALLOCATION AND SWAPPING

In general, a reconfigurable application exploits partial reconfiguration for three main tasks: area efficiency, power saving and reliability. In fact, the possibility to modify the content of the module and to implement a different functionality is useful to time-multiplex the area and to virtually have more resources then the ones physically available. On the other hand, the possibility to erase and eventually rewrite the content of a portion of configuration memory allows to have a system which power consumption scales with the payload, and it is fully programmed only when the maximum effort is needed. Additionally, as mentioned previously, with the refresh of the configuration memory it is possible to avoid or recover application failures. Except for the last purpose where also unused frames should be refreshed, the capillary reconfiguration can be used as strategy. In fact, when modules are allocated if needed and deallocated if in idle to perform power saving (e.g., [37], [38]), only the used frames are involved in the reconfiguration. When different hardware tasks need to be allocated in the same reconfigurable region to perform space/time partitioning and area saving, the situation is different. If blanking configurations are not used, the frames involved in the swap between two hardware tasks in a given reconfigurable region are all those frames that are

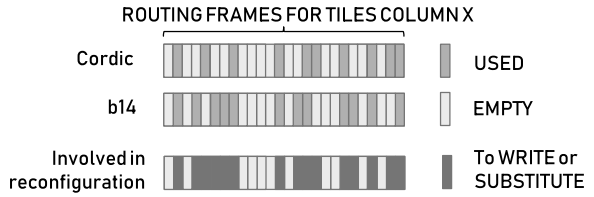


**FIGURE 15.** Example of Frames overlapping in routing frames for one Tiles column.

used in both designs, as shown in Figure 15. Since just the frames left to zero in both designs can be excluded from the procedure, smaller is the number of used frames, higher is the probability that empty frames coincide.

In order to confirm this, we made an additional analysis on the benchmarks to extend the evaluation to the case where, besides being allocated or deallocated, a swap between two designs must be performed in the same reconfigurable region. To perform a fair analysis, we coupled them according to their size and we made a comparison among the frames involved in the reconfiguration for the two situations with standard and FeDRA routing.

The results of this analysis are shown in Table 5. It is possible to see that with the standard routing, when modules swapping is required, a higher number of frames is considered. Anyway, when FeDRA optimization is applied, frame saving is achieved also in this scenario. As it is possible to see from the last column of Table 5, the frame saving goes from more than 37.62% for smaller circuits, to the 32.05% for the biggest, following the same trend of previous analysis.

**TABLE 5.** Frames involved for allocation/deallocation and Swapping of benchmark hardware tasks with Standard and FeDRA routing.

| Task and Banchmarks | Standard | FeDRA | Saving [%] |
|---|---|---|---|
| Allocate/Deallocate b05 | 143 | 87 | |
| Allocate/Deallocate b12 | 170 | 104 | |
| Swap b05 and b12 | 210 | 131 | **37.2%** |
| Allocate/Deallocate Cordic | 335 | 208 | |
| Allocate/Deallocate b14 | 442 | 291 | |
| Swap Cordic and b14 | 466 | 307 | **34.12%** |
| Allocate/Deallocate miniMIPS | 1,409 | 912 | |
| Allocate/Deallocate b22 | 1,145 | 764 | |
| Swap miniMIPS and b22 | 1,590 | 1,049 | **34.03%** |
| Allocate/Deallocate b18 | 3,185 | 2,108 | |
| Allocate/Deallocate b19 | 5,817 | 3,942 | |
| Swap b18 and b19 | 5,888 | 4,001 | **32.05%** |

### D. DISCUSSION

The result obtained thanks to FeDRA optimization demonstrates its efficiency. The link between programmable logic and configuration memory is a key point for the optimization of reconfigurable applications. This is generally true for

FPGA designs, since the low-level awareness can be a powerful tool for implementing fine-grained optimization and reliability techniques. In the reconfigurable scenario this becomes even more crucial, considering that the configuration memory becomes part of the application itself. Unfortunately, the current FPGA design trend goes in the direction of providing higher abstraction to the users. The lack of information about low-level resources organization from vendors makes challenging the development of efficient reconfigurable architectures. Another problem is that the body of knowledge and the tools for reconfigurable computing is scattered rather than unified. The analysis we performed on the configuration memory organization, bitstream encoding and its connection to resources allowed us to have the intuition behind the key idea of the proposed optimization. In fact, directly acting on the frames we are able to play at a deeper level, having a relevant reconfiguration time reduction without any appreciable impact on the programmable resource layer topology and performance.

## VII. CONCLUSION AND FUTURE WORKS

In this paper, we discuss a novel approach to reduce the reconfiguration time by reducing the number of configured frames of the FPGA interconnection network. Experimental results on a various set of benchmark circuits show an average reduction of reconfiguration time of 35% with respect to the traditional workflow, showing a negligible degradation even for extremely large circuits, thigh area constraints and in the case run time modules relocation.

As future work, we will perform evaluations for further improving our algorithm: to investigate the effect of the nets routing order on the frame saving; to make selective optimizations with FeDRA, by living static nets and critical paths routed by the vendor; and the possibility to cross-correlate the routing of different circuits to further reduce the reconfigured frames in the case of modules swapping in the same reconfigurable region. Additionally, we plan to make a comparison among standard and FeDRA approaches in terms of power consumption of the rerouted nets. Finally, we are currently studying the UltraScale Xilinx architecture, in order to extend the FeDRA optimization to other FPGA families.

## REFERENCES

[1] W. A. Najjar and P. Ienne, "Reconfigurable computing," *IEEE Micro*, vol. 34, no. 1, pp. 4–6, Jan./Feb. 2014.

[2] T. El-Ghazawi, E. El-Araby, M. Huang, K. Gaj, V. Kindratenko, and D. Buell, "The promise of high-performance reconfigurable computing," *Computer*, vol. 41, no. 2, pp. 67–76, Feb. 2008.

[3] M. Wirthlin, "High-reliability FPGA-based systems: Space, high-energy physics, and beyond," *Proc. IEEE*, vol. 103, no. 3, pp. 379–389, Mar. 2015.

[4] H.-M. Pham, S. Pillement, and S. J. Piestrak, "Low-overhead fault-tolerance technique for a dynamically reconfigurable softcore processor," *IEEE Trans. Comput.*, vol. 62, no. 6, pp. 1179–1192, Jun. 2013.

[5] H. Tan and R. F. DeMara, "A multilayer framework supporting autonomous run-time partial reconfiguration," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 16, no. 5, pp. 504–516, May 2008.

[6] L. Sterpone, M. Porrmann, and J. Hagemeyer, "A novel fault tolerant and runtime reconfigurable platform for satellite payload processing," *IEEE Trans. Comput.*, vol. 62, no. 8, pp. 1508–1525, Aug. 2013.

[7] P. Sedcole, B. Blodget, T. Becker, J. Anderson, and P. Lysaght, "Modular dynamic reconfiguration in Virtex FPGAs," *IEE Proc.-Comput. Digit. Techn.*, vol. 153, no. 3, pp. 157–164, 2006.

[8] *7 Series FPGAs Configuration User Guide UG470 (V1.13.1)*, Aug. 2018.

[9] J. Heiner, N. Collins, and M. Wirthlin, "Fault tolerant ICAP controller for high-reliable internal scrubbing," in *Proc. IEEE Aerosp. Conf.*, Mar. 2008, pp. 1–10.

[10] K. Vipin and S. A. Fahmy, "ZyCAP: Efficient partial reconfiguration management on the Xilinx Zynq," *IEEE Embedded Syst. Lett.*, vol. 6, no. 3, pp. 41–44, Sep. 2014.

[11] W. Guohua, L. Dongming, W. Fengzhou, A. Adetomi, and T. Arslan, "A tiny and multifunctional ICAP controller for dynamic partial reconfiguration system," in *Proc. NASA/ESA Conf. Adapt. Hardw. Syst. (AHS)*, Pasadena, CA, USA, Jul. 2017, pp. 71–76.

[12] E. Vansteenkiste, B. A. Farisi, K. Bruneel, and D. Stroobandt, "TPaR: Place and route tools for the dynamic reconfiguration of the FPGA's interconnect network," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 33, no. 3, pp. 370–383, Mar. 2014.

[13] L. Sterpone and L. Bozzoli, "Fast partial reconfiguration on SRAM-based FPGAs: A frame-driven routing approach," in *Proc. 14th Int. Symp. Appl. Reconfigurable Comput.*, 2018, pp. 319–330.

[14] P. Pfeifer, F. Hosseinzadeh, and H. T. Vierhaus, "On comparison of configurable encoders in Xilinx and Altera FPGAs," in *Proc. Int. Conf. Appl. Electron. (AE)*, Pilsen, Czech Republic, Sep. 2017, pp. 1–4.

[15] C. Gonzalez-Concejero, V. Rodellar, A. Alvarez-Marquina, E. M. D. Icaya, and P. Gomez-Vilda, "An FFT/IFFT design versus altera and xilinx cores," in *Proc. Int. Conf. Reconfigurable Comput. FPGAs*, Cancún, Mexico, Dec. 2008, pp. 337–342.

[16] *7 Series FPGAs Configurable Logic Block User Guide UG474 (V1.8)*, Sep. 2016.

[17] L. Bozzoli and L. Sterpone, "COMET: A configuration memory tool to analyze, visualize and manipulate FPGAs bitstream," in *Proc. ARCS Workshop 31th Int. Conf. Archit. Comput. Syst.*, Brunswick, Germany, 2018, pp. 1–4.

[18] M. Berg, C. Poivey, D. Petrick, D. Espinosa, A. Lesea, K. A. LaBel, M. Friendlich, H. Kim, and A. Phan, "Effectiveness of internal versus external SEU scrubbing mitigation strategies in a Xilinx FPGA: Design, test, and analysis," *IEEE Trans. Nucl. Sci.*, vol. 55, no. 4, pp. 2259–2266, Aug. 2008.

[19] G. L. Nazar, L. P. Santos, and L. Carro, "Fine-grained fast field-programmable gate array scrubbing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 5, pp. 893–904, May 2015.

[20] N. Abel, S. Manz, F. Grull, and U. Kebschull, "Increasing design changeability using dynamic partial reconfiguration," *IEEE Trans. Nucl. Sci.*, vol. 57, no. 2, pp. 602–609, Apr. 2010.

[21] K. Dang Pham, E. Horta, and D. Koch, "BITMAN: A tool and API for FPGA bitstream manipulations," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Mar. 2017, pp. 894–897.

[22] M. Gort and J. H. Anderson, "Reducing FPGA router run-time through algorithm and architecture," in *Proc. 21st Int. Conf. Field Program. Log. Appl.*, Sep. 2011, pp. 336–342.

[23] R. K. Korn, "An efficient variable-cost maze router," in *Proc. 19th Design Autom. Conf.*, 1982, pp. 425–431.

[24] K. Heyse, K. Bruneel, and D. Stroobandt, "Mapping logic to reconfigurable FPGA routing," in *Proc. 22nd Int. Conf. Field Program. Log. Appl. (FPL)*, Aug. 2012, pp. 315–321.

[25] C.-D. Chen, Y.-S. Lee, A. C.-H. Wu, and Y.-L. Lin, "TRACER-FPGA: A router for RAM-based FPGA's," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 14, no. 3, pp. 371–374, Mar. 1995.

[26] N.-W. Eum, T. Kim, and C.-M. Kyung, "CeRA: A router for symmetrical FPGAs based on exact routing density evaluation," *IEEE Trans. Comput.*, vol. 53, no. 7, pp. 829–842, Jul. 2004.

[27] L. Sterpone, M. S. Reorda, and M. Violante, "RoRA: A reliability-oriented place and route algorithm for SRAM-based FPGAs," in *Proc. Res. Microelectron. Electron., PhD*, Lausanne, Switzerland, vol. 1, Jul. 2005, pp. 173–176.

[28] *Vivado Design Suite User Guide Implementation UG904 (V2019.2)*, Dec. 2019.

[29] S. Hauck, A. Dehon, and M. Kaufmann, "Systems on silicon," in *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*. 2008.

[30] J. Resano, D. Mozos, and F. Catthoor, "A hybrid prefetch scheduling heuristic to minimize at run-time the reconfiguration overhead of dynamically reconfigurable hardware," in *Proc. Design, Automat. Test Eur.*, Mar. 2005, pp. 106–111.
[31] M. J. Wirthlin and B. L. Hutchings, "Sequencing run-time reconfigured hardware with software," in *Proc. 4th Int. ACM Symp. Field-Program. Gate Arrays*, Feb. 1996, pp. 122–128.
[32] G.-M. Wu, J.-M. Lin, and Y.-W. Chang, "Generic ILP-based approaches for time-multiplexed FPGA partitioning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.* vol. 20, no. 10, pp. 1266–1274, Oct. 2001.
[33] C. Beckhoff, D. Koch, and J. Torresen, "The xilinx design language (XDL): Tutorial and use cases," in *Proc. 6th Int. Workshop Reconfigurable Commun.-Centric Syst.-on-Chip (ReCoSoC)*, Montpellier, France, Jun. 2011, pp. 1–8.
[34] F. Corno, M. S. Reorda, and G. Squillero, "RT-level ITC'99 benchmarks and first ATPG results," *IEEE Des. Test. IEEE Des. Test. Comput.*, vol. 17, no. 3, pp. 44–53, May 2000.
[35] J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. Electron. Comput.*, vol. EC-8, no. 3, pp. 330–334, Sep. 1959.
[36] [Online]. Available: https://opencores.org/projects/minimips
[37] S. Liu, R. N. Pittman, and A. Forin, "Energy reduction with run-time partial reconfiguration," Microsoft Res., Redmond, WA, USA, Tech. Rep. MSR-TR-2009-2017, Sep. 2009.
[38] R. Tessier, S. Swaminathan, R. Ramaswamy, D. Goeckel, and W. Burleson, "A reconfigurable, power-efficient adaptive viterbi decoder," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, no. 4, pp. 484–488, Apr. 2005.

**LUDOVICA BOZZOLI** (Student Member, IEEE) received the B.S. degree from the Università di Roma Tre, Rome, Italy, and the M.S. degree from the Politecnico di Torino, Turin, Italy, in electronics engineering, where she is currently pursuing the Ph.D. degree with the Department of Computer and Control Engineering. Her main research topics include reconfigurable architectures and design techniques for their performance and dependability.

**LUCA STERPONE** (Member, IEEE) received the M.S. and Ph.D. degrees in computer engineering from the Politecnico di Torino, Italy, in 2003 and 2007, respectively. He is currently an Associate Professor with the Department of Computer Engineering, Politecnico di Torino. He is the author of more than 160 articles. His current research interests include reconfigurable computing, computer-aided design algorithms, fault tolerance architectures, and radiation effects on components and systems. He received several awards for his research activities.